МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ
імені адмірала Макарова

Г. В. Павлов, М. В. Покровський, І. Л. Вінниченко

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт з дисципліни „Комп'ютеризоване проектування цифрових електронних систем"

Рекомендовано Методичною радою НУК

Миколаїв ◊ НУК ◊ 2018

УДК 044.438(076)

ББК 32.844.1-022я73

П 12

Автори:

Г. В. Павлов, д-р техн. наук, професор;

М. В. Покровський, канд. техн. наук;

І. Л. Вінниченко

Рецензент О. К. Жук, канд. техн. наук, доцент

Павлов Г. В.

П 12 Методичні вказівки до виконання лабораторних робіт з дисципліни „Комп’ютеризоване проектування цифрових електронних систем” / Г. В. Павлов, М. В. Покровський, І. Л. Вінниченко – Миколаїв : НУК, 2018. – 32 с.

Подано теоретичні відомості та методичні вказівки до виконання лабораторних робіт з метою закріплення теоретичних знань основних принципів побудови і функціонування програмованих логічних інтегральних схем та опанування навиків та опанування практичних навиків написання програм на мові опису обладнання та апаратного забезпечення.

Призначено для студентів денної і заочної форми навчання спеціальності «Автоматизація та комп’ютерно-інтегровані технології». Можуть бути використані студентами інших спеціальностей, які вивчають дисципліну.

ВСТУП

Метою виконання лабораторних робіт є закріплення теоретичних знань основних принципів побудови і функціонування програмованих логічних інтегральних схем (ПЛІС), використання набутих практичних навиків написання програм на мові опису обладнання та апаратного забезпечення, а також вивчення програмних та апаратних засобів налагодження програм. У методичних вказівках наведено варіанти завдань для 8 лабораторних робіт, що охоплюють відомості з основних методів синтезу цифрових електронних систем (ЦЕС), розробки типових елементів, а саме:

1. Синтез ЦЕС на базі структурних та поведінкових моделей;
2. Розробка суматора;
3. Розробка мультиплікатора;
4. Розробка контролера динамічної оперативної пам’яті;
5. Розробка ядра *RISC*-процесора;
6. Розробка випробувального стенду для тестування *RISC*-процесора.

В кожній роботі наведено мету, завдання, теоретичні відомості та контрольні запитання. Лабораторні роботи проводяться з використанням середовищ *Active-HDL* та *ISE WebPACK*. Передбачається використання плати *Spartan* 3 *Starter Kit*.

Матеріали даних методичних вказівок також може бути використано при курсовому та дипломному проектуванні.

ЛАБОРАТОРНА РОБОТА № 1

Тема: Методологія проектування ЦЕС з застосуванням мови *Verilog*.

Мета:

1. Отримати знання о типах моделей, які описують ЦЕС;
2. Розглянути процес структурної декомпозиції.

Завдання:

1. Скласти інтерфейсні моделі помножувача/накопичувача комплексних чисел (ПНКЧ) з різними ступенями абстракції;
2. Скласти відповідні функціональні моделі ПНКЧ.

Теоретичні відомості

Методологія проектування ЦЕС включає кілька етапів. Спочатку формулюються технічні та експлуатаційні вимоги до верхнього рівня проектованого пристрою, а також визначається його інтерфейсна модель, що включає перелік входів і виходів, часові характеристики роботи пристрою. Згодом здійснюється структурна декомпозиція пристрою верхнього рівня, тобто розбиття його на складові компоненти, які виділяються за функціональними або топологічними міркуваннями. Отримані структурні компоненти в свою чергу можуть бути розділені на більш дрібні складові або реалізовані операторами мови *Verilog* на поведінковому рівні. Задача розбиття розроблювального пристрою на структурні складові є досить складною і визначає якість та ефективність всього процесу проектування надалі. Правильно здійснена структурна декомпозиція дозволяє значно спростити взаємодію між окремими компонентами проекту [1].

У процесі проектування ЦЕС на всіх його етапах створюється ряд моделей різних аспектів роботи цифрових пристроїв та різних рівнів абстракції. Причому процес проектування полягає в переході від моделей з вищим рівнем абстракції до моделей з нижчим рівнем абстракції, тобто

поступовій деталізації описання ЦЕС. Наведемо класифікацію моделей, що застосовуються в процесі проектування [1]:

1. Поведінкова модель показує реакцію моделі на зміну вхідних сигналів з урахуванням часу реакції, але не містить детального опису апаратної реалізації ЦЕС;
2. Функціональна модель описує функції системи без зазначення способу реалізації цих функцій;
3. Структурна модель представляє компоненти чи системи з погляду взаємозв’язків між елементами, що їх утворюють;
4. Модель продуктивності дозволяє визначити лише часові аспекти роботи ЦЕС, що моделюється;
5. Інтерфейсна модель містить деталізацію по всіх аспектах процесів обміну інформацією між об’єктом та зовнішнім середовищем, включаючи функціональність, часові характеристики, значення даних тощо.

На першому етапі проектування формуються інтерфейсна, продуктивна та функціональна моделі, із застосуванням яких можна постійно перевіряти моделі, сформовані на наступних етапах на відповідність технічному завданню. На другому етапі формується поведінкова модель. Під поведінковою розуміється модель, написана із застосуванням всіх існуючих в цій мові конструкцій та типів даних, наприклад, дійсних чисел, файлів, вказівників динамічної пам’яті тощо. Поведінкова модель, розроблена на другому етапі, повинна повністю відповідати вимогам, сформованим на першому етапі. Після формування остаточного вигляду поведінкової моделі створюється синтезна модель. Синтезна модель також належить до класу поведінкових, однак може бути написана лише за допомогою певної підмножини конструкцій мови опису обладнання та апаратного забезпечення, які підтримуються засобами синтезу логічної структури. На четвертому етапі здійснюється перехід від

синтезної моделі до логічної структури та бітового потоку, який завантажується безпосередньо в ПЛІС.

ПНКЧ є типовим пристроєм в системах цифрової обробки сигналів [2]. Цей пристрій має 2 комплексних входи $\dot{A}$ і $\dot{B}$, а також комплексний вихід $\dot{R}$, що являє собою суму добутків вхідних сигналів за певний проміжок часу. На найбільш високому і віддаленому від реалізації рівні абстракції алгоритм функціонування ПНКЧ можна записати в наступному вигляді:

$$\dot{R}(\tau) = \int_0^{\tau} \left(\dot{A}(t) \times \dot{B}(t)\right) \cdot dt .$$

Але ЦЕС не можуть реалізовувати ідеальний інтегратор, оскільки обчислювальні процеси є дискретними та виконуються за певний проміжок часу. Також мова *Verilog* не має спеціального типу даних для обробки комплексних чисел [2]. Таким чином, наступні моделі повинні враховувати період дискретизації та представлення комплексного числа парою дійсних чисел, які являють собою дійсну та мниму частину комплексного числа. Засоби синтезу логічних кіл в мові *Verilog* не підтримують роботи з дійсними числами. Тобто для подальшої деталізації моделі ПНКЧ необхідно представити дійсні числа наборами цілих чисел. Можна використати форму чисел з плаваючою комою, в якій число зберігається у вигляді мантиси і показника степеня. При цьому необхідно завдати кількість бітів для представлення мантиси та показника степеня. Також потрібно ввести до моделі додаткові сигнали управління – сигнал скидання ПНКЧ в початковий стан, сигнал синхронізації для визначення моментів часу, коли треба отримати вхідні дані, а також сигнал готовності результату обчислень.

Таким чином, для опису ПНКЧ необхідно побудувати 3 пари моделей. Кожна пара моделей (функціональна та інтерфейсна моделі) буде

описувати роботу ПНКЧ на певному рівні абстракції. При цьому моделі вищого рівня є джерелами еталонних вихідних сигналів у процесі тестування моделей нижчого рівня.

Контрольні запитання

1. Для чого використовується мова опису обладнання та апаратного забезпечення?
2. Наведіть приклади мов опису обладнання та апаратного забезпечення?
3. Які моделі використовуються для опису ЦЕС?
4. Назвіть ознаки поведінкової та функціональної моделей.
5. Назвіть ознаки структурної та інтерфейсної моделей.
6. У чому полягає процес структурної декомпозиції?

ЛАБОРАТОРНА РОБОТА № 2

Тема: Синтез ЦЕС на базі структурних *Verilog*-моделей.

Мета:

1. Оволодіти знаннями по проектуванню пристроїв на базі структурних моделей;
2. Навчитись будувати ЦЕС на мові *Verilog* за допомогою стандартних логічних елементів.

Завдання:

1. Описати на мові *Verilog* схему, яку наведено на рис. 2.1;
2. Провести моделювання наведеної схеми.

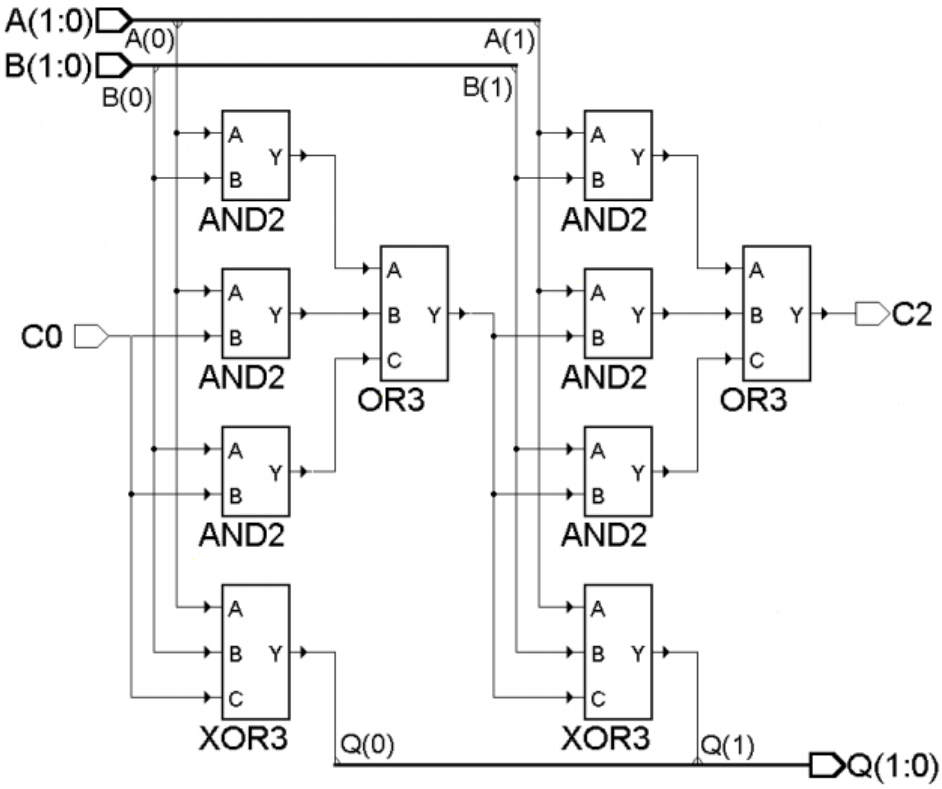

Рис. 2.1. Структура двохрозрядного суматора

Теоретичні відомості

Основною структурною одиницею устрою, що описується за допомогою мови опису апаратури, є модуль [2]. Всі модулі є незалежними об'єктами рівня проекту і не можуть включати всередині себе опису інших модулів. Але модуль може містити всередині себе посилання на інші модулі. Для розробки великих програм зручно розміщувати модулі в різних файлах в межах одного проекту.

У загальному випадку модуль має наступну структуру:

module Ім'я_модуля (Інтерфейс_модуля);

Опісаніе_інтерфейсу

...

Внутрішня_реалізація_модуля

...

endmodule

module, endmodule – зарезервовані слова, що означають початок і кінець модуля. Імя_модуля – ідентифікатор, за яким можна буде звернутися до даному модулю з інших частин програми. Інтерфейс_модуля – перелік вхідних і вихідних сигналів модуля, при цьому вказівки типу та напрямку немає. Опіс_інтерфейсу – перелічуються тип та напрямки сигналів, які описано в інтерфейсі модуля. Внутрішня_реалізація_модуля – це послідовність операторів мови *Verilog*, яка описує поведінку даного модуля.

Структурний опис архітектури представляє структуру об'єкта як композицію з компонентів, які з'єднані між собою і обмінюються сигналами [4]. Функції, що реалізуються компонентами, в явному вигляді не вказуються. Структурний опис включає імена і типи компонентів, з яких складається схема, а також їх зв'язки.

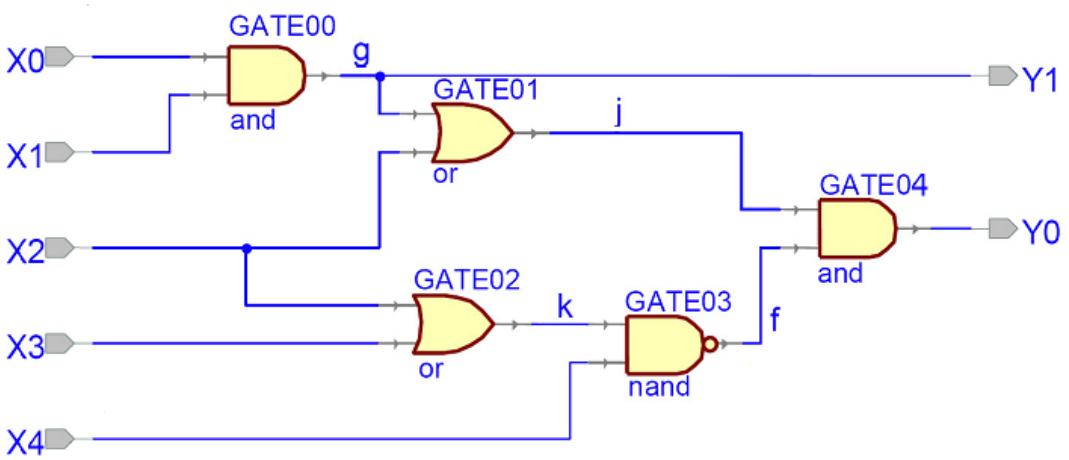

Рис. 2.2. Приклад комбінаційної схеми

Список стандартних логічних елементів мови *Verilog* містить: елемент І (*and*), елемент І-НЕ (*nand*), елемент АБО (*or*), елемент АБО-НЕ (*nor*), елемент ВИКЛЮЧНЕ АБО (*xor*), елемент ВИКЛЮЧНЕ АБО-НЕ (*xnor*) [4]. На відміну від модулів, що створюються користувачем, стандартні логічні елементи не мають фіксованої кількості вхідних портів. На вхід всіх перелічених елементів можна подавати довільну кількість вхідних сигналів, але при цьому максимальна кількість обмежується

програмою синтезу. Окрім перерахованих вище примітивів мова *Verilog* містить шість базових логічних примітивів: *buf*, *not*, *bufif*1, *bufif*0, *notif*1, *notif*0. Перши два являють собою стандартний буфер та інвертор, а решта – буфери та інвертори с додатковим входом дозволу видачі вихідного сигналу [4].

Розглянемо розробку пристрою на логічному рівні абстракції на прикладі схеми, що наведено на рис. 2.2. *Verilog*-опис наведеної схеми може бути реалізоване на вентильному рівні за допомогою стандартних логічних елементів у вигляді модуля *comb_net*:

```
module comb_net (X0, X1, X2, X3, X4, Y0, Y1);
// Опис портів и внутрішніх сигналів:
 input X0, X1, X2, X3, X4;          // Вхідні сигнали
 output Y0, Y1;                     // Вихідні сигнали
wire X0, X1, X2, X3, X4, Y0, Y1;
wire k , j, f, g;                   // Внутрішні сигнали
//Опис внутрішньої структури:
// Включення стандартного елемента and під іменем GATE00:
and GATE00 (Y1, X0, X1);
// Включення решти стандартних елементів:
or GATE01 (j, Y1, X2);
or GATE02 (k, X2, X3);
nand GATE03 (f, k, X4);
and GATE04 (Y0, f, j);
endmodule
```

Після успішної компіляції одержану програму моделюють після виклику вікна графіків сигналів. В цьому вікні встановлюють імена сигналів, які необхідно відобразити, в тому числі всі вхідні сигнали. Всі вхідні сигнали необхідно підключити до стимуляторів. Серед стимуляторів можна вибрати генератор синхросигналів з заданим періодом, лічильник

або задану послідовність зміни сигналу. Одержані графіки сигналів заносять в протокол лабораторної роботи.

Контрольні запитання

1. Опішить структуру модуля на мові *Verilog*.
2. Характерні ознаки структурного опису?
3. Назвіть стандартні логічні елементи.
4. Наведіть базові логічні примітиви.
5. Скільки входів мають стандартні логічні елементи?
6. Як відбувається включення до проекту стандартного елемента або раніше описаного модулю?

ЛАБОРАТОРНА РОБОТА № 3

Тема: Синтез ЦЕС на базі поведінкових *Verilog*-моделей.

Мета:

1. Оволодіти знаннями по проектуванню пристроїв на базі поведінкових моделей;
2. Навчитись будувати послідовні вузли ЦЕС на мові *Verilog*.

Завдання:

1. Розробити програму лічильника імпульсів з інтерфейсом, який наведено у табл. 3.1;
2. Провести моделювання розробленої програми, здійснити синтез;
3. З використанням плати *Spartan* 3 *Starter Kit* розробити пристрій, що лічить кількість натисків на кнопку в десятковій системі та виводить одержане число на два розряди світлодіодного індикатора.

Таблиця 3.1 Інтерфейс лічильника

Назва	**Опис**
Входи	
CLK	Сигнал синхронізації
CE	Сигнал, що дозволяє лічити імпульси
RST	Сигнал синхронного скидання лічильника
Виходи	
CO	Сигнал переповнення розрядної сітки лічильника
Q[3:0]	Вихідний сигнал лічильника (кількість імпульсів), беззнакове ціле число, діапазон значень від 0 до 15 (десяткова система)
Константи	
Max_Val	Максимальне значення вихідного сигналу *Q*[3:0], беззнакове ціле число, діапазон значень від 1 до 15 (десяткова система)

Теоретичні відомості

Поведінковий опис пристрою – це опис залежностей його стану (вихідних сигналів) від вхідних сигналів і від попереднього стану. Інакше кажучи, поведінковий опис задає функцію або алгоритм, реалізований пристроєм. Поведінковий опис може мати наступні форми:

1. Опис необхідного логічного перетворення таблицею функціонування (таблицею істинності), для опису автоматів з пам'яттю іноді використовується представлення таблиць функціонування в формах карт Карно;
2. Аналітичний опис логічного перетворення, яке повинно здійснюватися проектованим пристроєм (логічні вирази), наприклад, досконала диз'юнктивна нормальна форма;
3. Опис необхідного логічного перетворення за допомогою часових діаграм сигналів на входах і виходах об'єкта;
4. Опис об'єкта діаграмою станів;

5. Текстова форма, яка, як і при структурному описі, може являти собою текст на одній з мов опису апаратури (*Verilog*, *VHDL*).

Вузли цифрових пристроїв підрозділяють на комбінаційні та послідовні. Комбінаційні вузли називають також автоматами без пам'яті. Послідовні вузли називають автоматами з пам'яттю. Значення вихідних сигналів комбінаційного вузла залежать тільки від поточних значень вхідних сигналів. На відміну від цього, значення вихідних сигналів послідовного вузла (стан послідовного вузла) залежать не тільки від значень сигналів, що надходять на їх входи, але й від того стану, який передував подачі цих вхідних сигналів і визначався раніше діючими вхідними сигналами (передісторією). Зазначені функціональні (поведінкові) відмінності пов'язані зі структурними відмінностями між комбінаційними та послідовними вузлами: для послідовних вузлів характерні внутрішні зворотні зв'язки, а в комбінаційних вузлах такі зворотні зв'язки відсутні.

До послідовних пристроїв відносяться тригери, лічильники, регістри. Лічильником називається пристрій, який призначено для підрахунку вхідних імпульсів в тому чи іншому коді. Лічильник являє собою зв'язане коло тригерів, що утворюють пам'ять із заданим числом стійких станів. Класифікація лічильників: за напрямом (інкрементні, декрементні та реверсивні), за способом кодування (позиційні, непозиційні), за способом міжрозрядних зв’язків (синхронні, асинхронні), за модулем (двійкові, десяткові).

Робота лічильника з інтерфейсом, який наведено у табл. 3.1, описується наступним чином. Лічильник збільшує значення вихідного сигналу *Q* кожного разу, коли значення сигналу синхронізації *CLK* змінюється з 0 на 1 та сигнал дозволу *CE* дорівнює 1. При досягненні лічильником максимального значення *Max_Val* відлік починається спочатку, одночасно вихід *CO* набуває значення 1, яке утримується

впродовж одного такту сигналу синхронізації. Якщо на вході скидання *RST* буде присутній сигнал зі значенням 1, то при зміні сигналу синхронізації *CLK* з 0 на 1 відбувається скидання в 0 всіх вихідних сигналів.

Контрольні запитання

1. Що описує поведінкова модель пристрою?
2. Які форми може мати поведінковий опис?
3. Чим відрізняються комбінаційні та послідовні пристрої?
4. Що відноситься до послідовних пристроїв?
5. Що таке лічильник?
6. Чим відрізняються синхронні та асинхронні лічильники?

ЛАБОРАТОРНА РОБОТА № 4

Тема: Розробка суматора.

Мета:

1. Отримати знання щодо форматів представлення даних в арифметичних операціях;
2. Отримати навики проектування суматорів з використанням поведінкової моделі.

Завдання:

1. Розробити програму 8-розрядного суматора цілих беззнакових чисел з інтерфейсом, який наведено у табл. 4.1;
2. Провести моделювання розробленої програми;
3. Модифікувати раніше розроблену програму для обробки цілих знакових чисел;
4. Провести моделювання модифікованої програми.

Таблиця 4.1 Інтерфейс суматора

Назва	**Опис**
Входи	
CLK	Сигнал синхронізації
RST	Сигнал синхронного скидання суматора
A(7:0), *B*(7:0)	Доданки
CI	Біт переносу з додавання молодших розрядів
Виходи	
CO	Біт переносу в старший розряд
S(7:0)	Сума

Теоретичні відомості

В цифрових пристроях інформація різного характеру представляється двійковими послідовностями з різним числом розрядів. Окремі частини цих послідовностей мають певні призначення. В залежності від типу інформації використовується той чи інший формат даних. Формати даних можна розбити на дві групи: цілочисельні формати та формати для чисел із плаваючою комою. Цілочисельні формати використовуються для представлення цілих чисел і інформації, що може бути закодована такими числами. Формат із плаваючою комою використовується для представлення дробових або дійсних чисел [5]. Цілочисельні формати в свою чергу можна поділити на знакові та беззнакові. В цілому беззнаковому числі всі розряди використовуються для кодування значення. В знаковому числі один з розрядів (як правило, старший) використовується для формування знаку числа, а решта – для кодування значення. Основною характеристикою цілочисельних форматів при виконанні арифметичних операцій, є діапазон представлень чисел (*N*),

що прямо залежить від розрядності формату (b): для беззнакових форматів $0 \leq N \leq 2^b-1$, для форматів з врахуванням знаку: $-2^{b-1} \leq N \leq 2^{b-1}-1$.

Знакові цілі числа можуть бути представлені в зворотному або додатковому коді. Зворотний код для перетворення позитивного числа в негативне та навпаки використовує операцію інверсії. Але у такому випадку числу 0 відповідають 2 коди – „позитивний” 0 та „негативний” 0. Це призводить к ускладненню арифметичних операцій. Додатковий код для зміни знаку числа використовує інверсію та додавання 1 до результату інверсії. В обох кодах старший розряд числа представляє знак. Якщо старший розряд дорівнює 0, то це позитивне число, а якщо 1 – негативне.

Суматор – це схема, яка призначена для підсумовування двох вхідних двійкових n-розрядних кодів. Операція віднімання замінюється складанням у зворотному або додатковому коді. Операції множення і ділення зводяться до реалізації багаторазових складань і зрушень. Тому суматор є важливим компонентом цифрових пристроїв. Для можливості каскадного з’єднання суматорів використовуються біти переносу, які забезпечують облік результатів додавання молодших частин числа у результатах додавання старших частин.

Контрольні запитання

1. Які формати даних існують?
2. Чим відрізняються цілочисельні формати та формати із плаваючою комою?
3. Наведіть приклади позитивних та негативних чисел у зворотному коді.
4. Наведіть приклади позитивних та негативних чисел у додатковому коді.
5. Що таке суматор?
6. Як можна замінити операцію віднімання операцією додавання?
7. Для чого використовуються біти переносу у суматорах?

ЛАБОРАТОРНА РОБОТА № 5

Тема: Розробка мультиплікатора.

Мета:

1. Отримати знання щодо форматів представлення даних в арифметичних операціях;
2. Отримати навики проектування мультиплікаторів з використанням поведінкової моделі.

Завдання:

1. Розробити програму 8-розрядного мультиплікатора цілих беззнакових чисел з інтерфейсом, який наведено у табл. 5.1;
2. Провести моделювання розробленої програми;
3. Модифікувати раніше розроблену програму для обробки 8-розрядних знакових чисел в форматі з фіксованою комою, кома після знакового розряду;
4. Провести моделювання модифікованої програми.

Таблиця 5.1 Інтерфейс мультиплікатора

Назва	**Опис**
Входи	
CLK	Сигнал синхронізації
RST	Сигнал синхронного скидання мультиплікатора
A(7:0), *B*(7:0)	Множники
Виходи	
OV	Біт переповнення
P(7:0)	Добуток

Теоретичні відомості

В цифрових пристроях інформація різного характеру представляється двійковими послідовностями з різним числом розрядів.

Окремі частини цих послідовностей мають певні призначення. В залежності від типу інформації використовується той чи інший формат даних. Формати даних можна розбити на дві групи: цілочисельні формати та формати для чисел із плаваючою комою. Цілочисельні формати використовуються для представлення цілих чисел і інформації, що може бути закодована такими числами. Формат із плаваючою комою використовується для представлення дробових або дійсних чисел. Цілочисельні формати в свою чергу можна поділити на знакові та беззнакові. В цілому беззнаковому числі всі розряди використовуються для кодування значення. В знаковому числі один з розрядів (як правило, старший) використовується для формування знаку числа, а решта – для кодування значення. Основною характеристикою цілочисельних форматів при виконанні арифметичних операцій, є діапазон представлень чисел (N), що прямо залежить від розрядності формату (b): для беззнакових форматів $0 \leq N \leq 2^b-1$, для форматів з врахуванням знаку: $-2^{b-1} \leq N \leq 2^{b-1}-1$ [3].

Знакові цілі числа можуть бути представлені в зворотному або додатковому коді. Зворотний код для перетворення позитивного числа в негативне та навпаки використовує операцію інверсії. Але у такому випадку числу 0 відповідають 2 коди – „позитивний” 0 та „негативний” 0. Це призводить к ускладненню арифметичних операцій. Додатковий код для зміни знаку числа використовує інверсію та додавання 1 до результату інверсії. В обох кодах старший розряд числа представляє знак. Якщо старший розряд дорівнює 0, то це позитивне число, а якщо 1 – негативне.

Суматор – це схема, яка призначена для підсумовування двох вхідних двійкових n-розрядних кодів. Операція віднімання замінюється складанням у зворотному або додатковому коді. Операції множення і ділення зводяться до реалізації багаторазових складань і зрушень. Тому суматор є важливим компонентом цифрових пристроїв. Для можливості каскадного з’єднання суматорів використовуються біти переносу, які

забезпечують облік результатів додавання молодших частин числа у результатах додавання старших частин.

Контрольні запитання

1. Які формати даних існують?
2. Чим відрізняються цілочисельні формати та формати із плаваючою комою?
3. Наведіть приклади позитивних та негативних чисел у зворотному коді.
4. Наведіть приклади позитивних та негативних чисел у додатковому коді.
5. Що таке суматор?
6. Як можна замінити операцію віднімання операцією додавання?
7. Для чого використовуються біти переносу у суматорах?

ЛАБОРАТОРНА РОБОТА № 6

Тема: Розробка контролера динамічної оперативної пам’яті.

Мета:

1. Отримати знання про функціонування динамічної оперативної пам’яті;
2. Отримати навики проектування контролера динамічної оперативної пам’яті.

Завдання:

1. Розробити програму контролера динамічної оперативної пам’яті з інтерфейсом, який наведено у табл. 6.1;
2. Провести моделювання розробленої програми.

Таблиця 6.1 Інтерфейс контролера динамічної оперативної пам’яті

Назва	**Опис**
Входи	
CLK	Сигнал синхронізації
RST	Сигнал синхронного скидання контролера динамічної оперативної пам’яті
RW	Напрям передачі інформації
AS	Готовність вхідної адреси
Addr_in(7:0)	Вхідна адреса комірки пам’яті
Виходи	
WE	Дозвіл запису
RAS	Активація адреси рядка
CAS	Активація адреси стовпця
ACK	Сигнал готовності даних
Addr_out(3:0)	Вихідна адресна шина

Теоретичні відомості

Динамічна оперативна пам’ять (*Dynamic Random Access Memory, DRAM*) є одним з видів пам'яті із довільним доступом. *DRAM* характеризується великою щільністю комірок, що забезпечує її низьку вартість [5]. Комірка *DRAM* являє собою конденсатор, заряд якого відповідає значенню біта інформації, що зберігається у комірці. Але цей заряд поступово зменшується та через деякий час інформація втрачається. Тому для *DRAM* характерним є періодичний процес регенерації, який оновлює заряди у комірках.

Для зменшення кількості виводів мікросхеми *DRAM* повна адреса комірки поділяється на адресу рядка та адресу стовпця, які передаються до мікросхеми *DRAM* послідовно. Для зазначення частини адреси, яка

передається у поточний момент, використовуються сигнали активації рядка та стовпця (*RAS* та *CAS*).

На рис. 6.1 наведена схема підключення контролера *DRAM*, який з'єднує процесор та мікросхему *DRAM*.

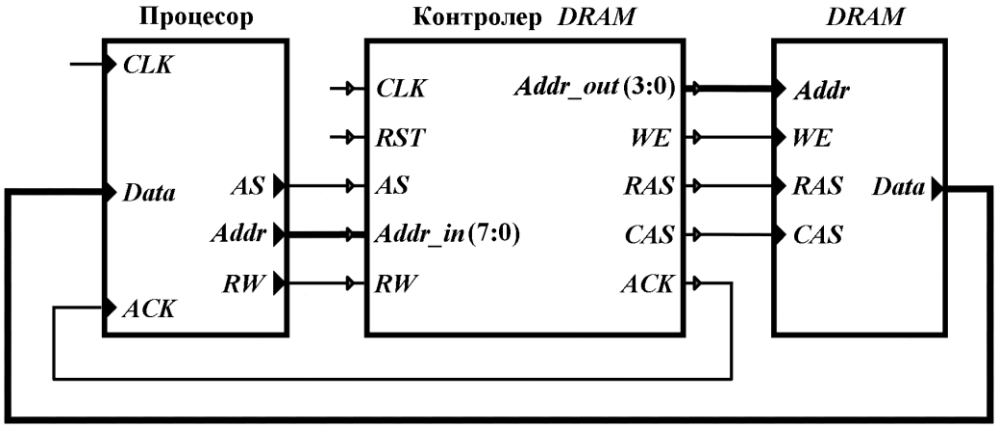

Рис. 6.1. Схема підключення контролера *DRAM*

Контролер DRAM приєднується до шини адреси і при виконанні операцій запису/зчитування здійснює перетворення 8-бітної адреси, що передається від процесора по шині *Addr_in*(7:0), в дві 4-бітних адреси – номер рядка та номер стовпця в матриці мікросхеми *DRAM*, які потім у послідовному режимі передаються мікросхемі *DRAM* по шині *Addr_out*(3:0). Крім того, завдання контролера *DRAM* полягає в періодичному генеруванні сигналів для проведення циклу регенерації. У разі несвоєчасного проведення циклу регенерації інформація у комірках мікросхеми *DRAM* буде втрачена. Часова діаграма функціонування контролера *DRAM* при здійсненні операції зчитування з мікросхеми *DRAM* наведена на рис. 6.2. На діаграмі можна виділити 5 станів контролера *DRAM*:

1. Очікування;
2. Вибір рядка;
3. Вибір стовпця;
4. Зчитування даних;
5. Готовність даних.

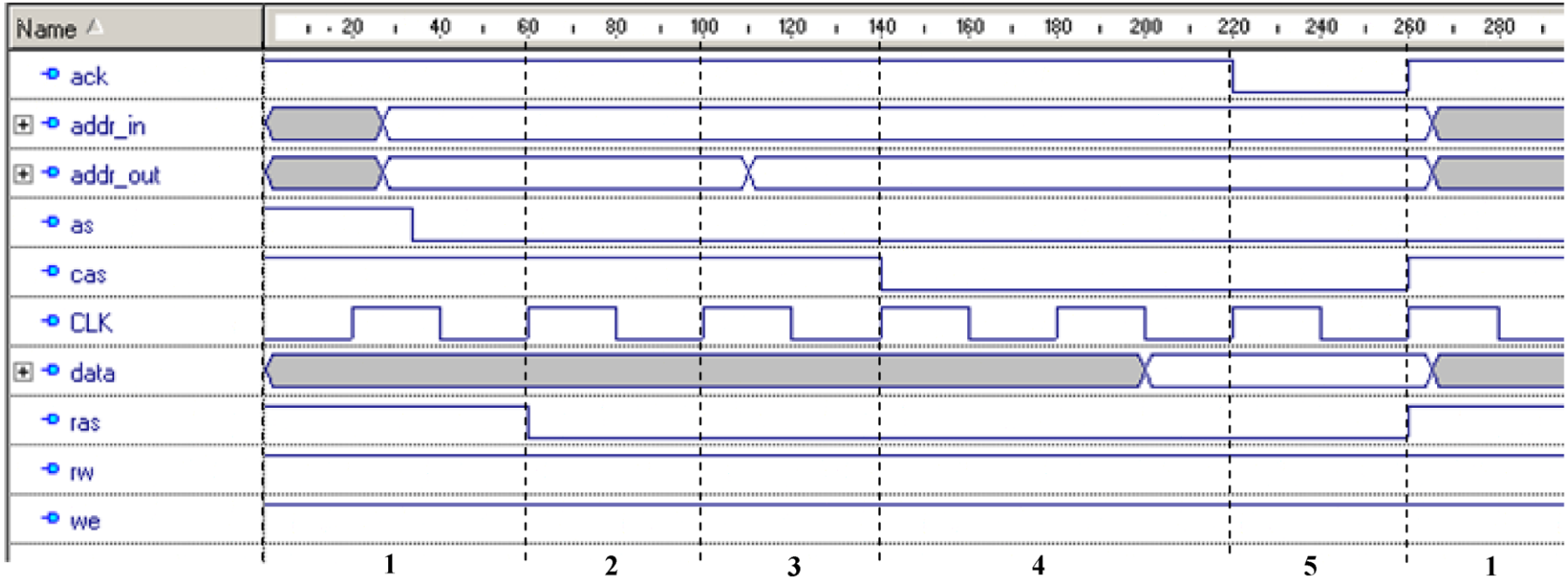

Рис. 6.2. Часова діаграма функціонування контролера *DRAM* при здійсненні операції зчитування

В процесі операції читання або запису спочатку на шину адреси подається адреса рядка. При активізації сигналу *RAS* вона завантажується в регістр адреси рядка мікросхеми *DRAM*. Через деякий час на шину адреси подається адреса стовпця, яка завантажується в регістр засувку адреси стовпця при активізації сигналу *CAS*. Далі в залежності від сигналу напряму дані або виставляються на шину даних, або зчитуються з неї.

Контрольні запитання

1. Опишіть принцип збереження даних у динамічної оперативної пам’яті.
2. Чим викликана поява режиму регенерації?
3. Функції сигналів *RAS* та *CAS*?
4. Функції сигналів *AS, RW, WE* та *ACK*?
5. Наведіть часову діаграму зчитування даних.
6. Наведіть часову діаграму запису даних.
7. Наведіть часову діаграму регенерації.

ЛАБОРАТОРНА РОБОТА № 7

Тема: Розробка ядра *RISC*-процесора.

Мета:

1. Отримати знання про функціонування *RISC*-процесора;
2. Отримати навики проектування ядра *RISC*-процесора.

Завдання:

1. Розробити програму ядра 4-бітного *RISC*-процесора з інтерфейсом, який наведено у табл. 7.1;
2. Забезпечити виконання ядром *RISC*-процесора інструкцій, які наведено у табл. 7.2.

Таблиця 7.1 Інтерфейс ядра *RISC*-процесора

Назва	**Опис**
Входи	
CLK	Сигнал синхронізації
RST	Сигнал синхронного скидання *RISC*-процесора
Виходи	
RAM(3:0)	Вихідна адресна шина пам’яті даних
ROM(6:0)	Вихідна адресна шина пам’яті програм
AO(3:0)	Стан акумулятору
CO	Прапор переносу
ZO	Прапор нульового результату
Входи/виходи	
Data(7:0)	Шина даних

Таблиця 7.2 Набір інструкцій *RISC*-процесора

Тип інструкції	Інструкція	Кодування	Опис
Переміщення	*load Rd*	*xxxx*0001	*ACC*:=[*xxxx*]
Переміщення	*load #const*	*xxxx*0011	*ACC*:=*xxxx*
Переміщення	*store Rd*	*xxxx*0101	[*xxxx*]:=*ACC*
Арифметична	*add Rd*	*xxxx*0111	*ACC*:=*ACC*+[*xxxx*]+*C*
Арифметична	*add #const*	*xxxx*1001	*ACC*:=*ACC*+*xxxx*+*C*
Логічна	*xor Rd*	*xxxx*1011	*ACC*:=*ACC xor* [*xxxx*]
Логічна	*test Rd*	*xxxx*1101	*ACC and* [*xxxx*]
Бітова	*clrc*	00001111	*C*:=0
Бітова	*setc*	00011111	*C*:=1
Переходу	*skipc*	00101111	*PC*:=*PC*+1+*C*
Переходу	*skipz*	00111111	*PC*:=*PC*+1+*Z*
Переходу	*jmp Addr*	*xxxxxxx*0	*PC*:=*xxxxxxx*

Теоретичні відомості

Одним з таких критеріїв класифікації процесорів є архітектура набору інструкцій процесора. Більшість сучасних процесорів побудовано на базі *CISC, RISC, VLIW* архітектур. *CISC* характеризується порівняно великою кількістю інструкцій, які мають різний час виконання та займають різну кількість пам'яті, що призводить до ускладнення пристрою декодування інструкцій та падінню продуктивності. *RISC* характеризується порівняно невеликою кількістю інструкцій з фіксованим розміром та фіксованим часом виконання. Це призводе до спрощення пристрою декодування інструкцій та збільшенню продуктивності. *VLIW* характеризується паралелізмом виконання певної кількості інструкцій, що забезпечує високу продуктивність при малих апаратних витратах. Але це можливо лише у випадку незалежності даних у відповідних інструкціях, що виконуються одночасно. Таким чином, *RISC* є найбільш простою архітектурою з точки зору реалізації процесора на базі ПЛІС.

В якості прикладу розглянемо модель 4-бітного процесору, який має 8-бітний формат інструкцій, з роздільними адресними просторами пам’яті програм та пам’яті даних. Процесор містить наступні основні вузли:

1. Постійну програмну пам’ять для 128 інструкцій, тобто ця пам’ять містить 8-бітні комірки з адресами від 0*x*00 до 0*x*7*F*;
2. Пам’ять даних з довільним доступом для 16 тетрад, тобто ця пам’ять містить 4-бітні комірки з адресами від 0*x*00 до 0*x*0*F*;
3. Пристрій обчислення адреси та завантаження інструкцій з пам’яті програм;
4. Пристрій декодування інструкцій;
5. Арифметико-логічний пристрій;
6. Акумулятор;
7. Прапори переносу і нульового результату.

Для обчислення адреси інструкцій використовується регістр програмного лічильника *PC*. Його розрядність відповідає максимальному об'єму пам'яті програм і складає 7 біт. При виконанні більшості інструкцій цей регістр збільшується на одиницю для завантаження наступної інструкції. При виконанні інструкцій переходу регістр *PC* змінюється відповідно табл. 7.2.

Після завантаження інструкція декодується та виконується. Для декодування використовується молодша тетрада інструкції. Але у випадку бітових інструкцій або інструкцій переходу правила декодування змінюються. Якщо молодший біт інструкції дорівнює нулю, то ця інструкція є інструкцією безумовного переходу та решта бітів містять адресу переходу. Бітові інструкції та інструкції умовного переходу мають однакові значення молодшої тетради, тому необхідно додатково аналізувати значення старшої тетради.

Інструкції переміщення, а також арифметичні та логічні інструкції виконуються з використанням акумулятору *ACC*. Інструкції переміщення та арифметичні інструкції можуть залучати як пам'ять даних, та і константи. А логічні інструкції можуть використовувати лише пам'ять даних. На стан прапорів переносу (*C*) та нульового результату (*Z*) впливає результат виконання лише арифметичних, логічних та бітових інструкцій.

Контрольні запитання

1. Порівняйте різні архітектури набору інструкцій процесора.
2. Які основні вузли містить процесор?
3. Як змінюється програмний лічильник при виконанні програми?
4. Перечисліть типи інструкцій, які реалізовано у лабораторній роботі.
5. Які інструкції впливають на стан прапорів?
6. Як відбувається декодування інструкцій?
7. Як стан прапорів впливає на виконання програми?

ЛАБОРАТОРНА РОБОТА № 8

Тема: Розробка випробувального стенду для тестування *RISC*-процесора.

Мета:

1. Отримати знання про тестування складних проектів;
2. Отримати навики розробки випробувального стенду для тестування *RISC*-процесора.

Завдання:

1. Розробити випробувального стенду для тестування ядра 4-бітного *RISC*-процесора з інтерфейсом, який наведено у табл. 7.1;
2. Провести моделювання виконання тестової програми, яку сформовано відповідно набору інструкцій у табл. 7.2.

Теоретичні відомості

Процес проектування цифрових пристроїв нерозривно пов'язаний з поняттям верифікації. Поняття верифікації в загальному випадку має на увазі під собою перевірку збігу результатів роботи пристрою, що розробляється, та вимог, які до них пред'являються. Верифікація може займати значну частину загального часу проектування. Більшість існуючих методик проектування цифрових пристроїв передбачають верифікацію лише на етапі налагодження готового пристрою. При такому підході навіть невелика помилка, занесена на ранній стадії, надалі може привести до серйозніших наслідків, що зазвичай виражається в додаткових матеріальних і часових витратах.

Функціональне тестування є невід'ємною частиною верифікації складних проектів. На цій стадії процесу симулюється поведінка спроектованого пристрою та перевіряється на відповідність вимогам, описаним у функціональній специфікації проекту [6]. Однією з метрик функціонального тестування є тестове покриття. Тестове покриття розглядається у двох основних аспектах: як покриття вимог і як кодове

покриття (тобто покриття компонентів коду). Якщо розглядати тестування як перевірку відповідності між реальним і очікуваною поведінкою пристрою, яка здійснюється при кінцевому наборі тестів, то саме ця безліч тестів і визначатиме тестове покриття. Чим вище повнота тестового покриття, тим більше тестів буде вибрано для перевірки тестованих вимог і коду.

Найпростішим рішенням забезпечення тестового покриття цифрового автомата з пам'яттю є перебір всіх можливих комбінацій вхідних сигналів [6]. Однак повний перебір вхідних комбінацій не гарантує відсутність помилок. Наприклад, відмова може виникати тільки у випадку специфічного поєднання внутрішніх станів і вхідних сигналів. Імовірність того, що подібне поєднання виникне внаслідок випадкової або псевдовипадкова генерації тестів, дуже низька. Щоб виникло потрібне поєднання, може знадобитися велика кількість тестів. Тому часто використовуються методи з використанням низки тверджень, на спрямованих на скорочення кількості тестових послідовностей при забезпеченні повного тестового покриття. Незважаючи на значне зменшення кількості тестових комбінацій в таких методах, ручне тестування часто залишається неможливим.

Автоматизація процесу верифікації виконується наступним чином:

1. Формується безліч вхідних і еталонних тестових комбінацій;
2. Проводиться запис наборів комбінацій в текстові файли;
3. На спроектований пристрій подаються вхідні тестові комбінації;
4. Проводиться симуляція роботи тестового блоку з подачею вихідних і еталонних комбінацій сигналів на схему порівняння;
5. Результат побітового порівняння двох масивів записується в звіт по верифікації.

Для тестування ядра 4-бітного *RISC*-процесора рекомендується програма, яка повинна виконувати складання двох 8-бітних чисел (наприклад, 0*x*29 + 0*x*48 = 0*x*71):

load #8
*store R*0
load #4
*store R*1
load #9
*store R*2
load #2
*store R*3
clrc
*load R*0
*add R*2
*store R*4
*load R*1
*add R*3
*store R*5

Loop:

*load R*4
*load R*5
jmp Loop

Для симуляції роботи наведеної програми необхідно перетворити інструкції у машинний код та забезпечити симуляцію розташування інструкцій у пам’яті програм процесора. Перетворення у машинний код виконується за допомогою табл. 7.2. Програма містить нескінчений цикл, тому немає необхідності додавати умову завершення симуляції. Після виконання програми у комірках *R*4 та *R*5 буде отримано результат.

Контрольні запитання

1. Як відбувається верифікація спроектованого пристрою?
2. Чим відрізняється детермінована та псевдовипадкова верифікація?
3. Що таке тестове покриття?
4. Які особливості має верифікація цифрового автомата з пам'яттю?
5. Які методи використовуються для зниження кількості тестових комбінацій?
6. Як відбувається автоматизація процесу верифікації?
7. Наведіть приклад тестової програми для верифікації інструкцій переміщення даних, арифметичних інструкцій та інструкцій переходу.

ЗМІСТ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. **Кондратенко Ю.П.** Verilog-HDL для моделирования и синтеза цифровых электронных схем: Учебное пособие / Ю.П. Кондратенко, В.В. Мохор, С.А. Сидоренко. – Николаев: Изд-во НФ НаУКМА, 2002. – 220 с.
2. **Кондратенко Ю.П.** VHDL-моделі для проектування цифрових пристроїв. Методичні вказівки до циклу лабораторних робіт / Ю.П. Кондратенко, С.А. Сидоренко, Д.М. Підопригора. – Миколаїв: УДМТУ, 2000. – 54 с.
3. **Мальцев П.П.** Программируемые логические ИМС на КМОП-структурах и их применение / П.П. Мальцев, Н.И. Гарбузов, А.П. Шарапов, Д.А. Кнышев. – М.: Энергоатомиздат, 1998. – 160 с.
4. Active-HDL User’s Guide. Second edition. – Copyright ALDEC, Inc. 1999. – 213 p.
5. **Bhasker J.** A VHDL synthesis primer. / J. Bhasker. – San Francisco: Morgan Kaufmann Publishers, 1996. – 688 p.
6. **Gray J.** Designing a simple FPGA-optimized RISC CPU and system-on-a-chip / J. Gray. – Bellevue: Gray research LLC, 2000. – 17 p.